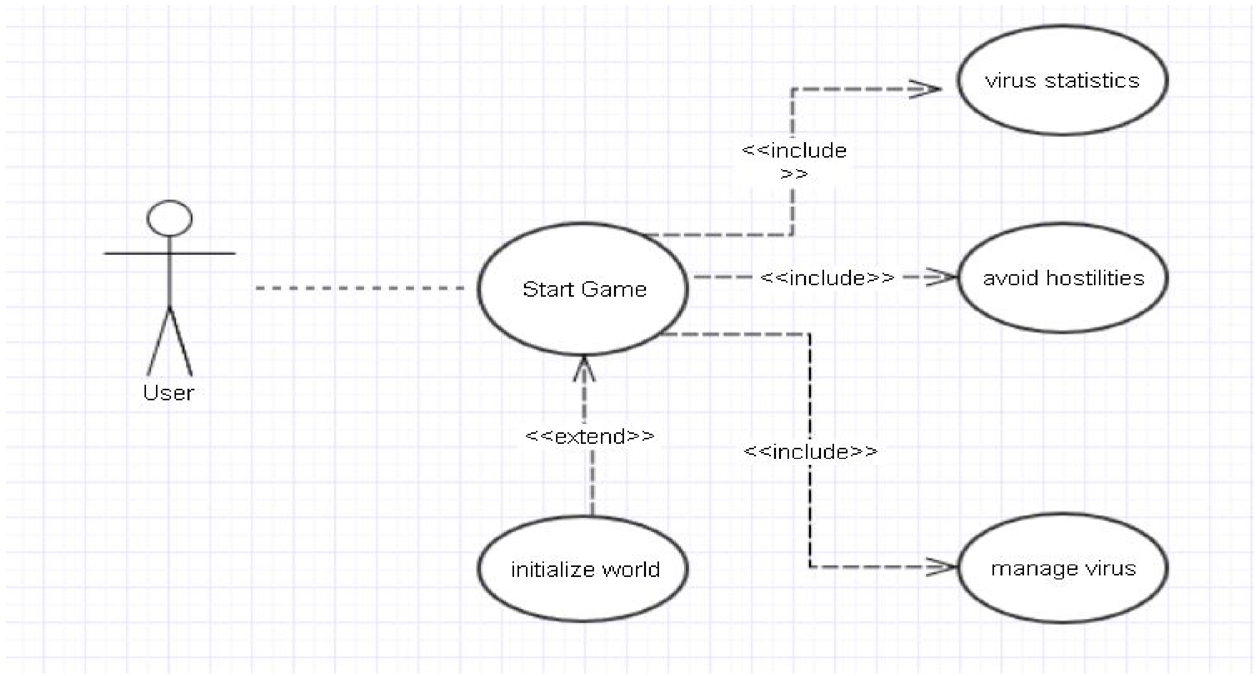
Software Specification for Virus Simulation

1. Introduction

1.1 Purpose

To observe the effects of a virus upon a population of people and understand the dynamics behind controlling how it spreads. Also, to make a fun, involved 2D video game that people will love to play!



This is our Use-Case Diagram. It shows how the user interacts with our program and the simple actions that it involves.

1.2 Intended Audience

The audience for this simulation/game is wide. Hopefully those who wish to study on how a virus can spread through a populace can find this simulation to be interesting to watch and vary how the virus infects. Also observing how the virus spreads through people moving around and coming into contact with others. For others it could just be a fun game seeing how they can infect people and seeing how they can make it spread faster.

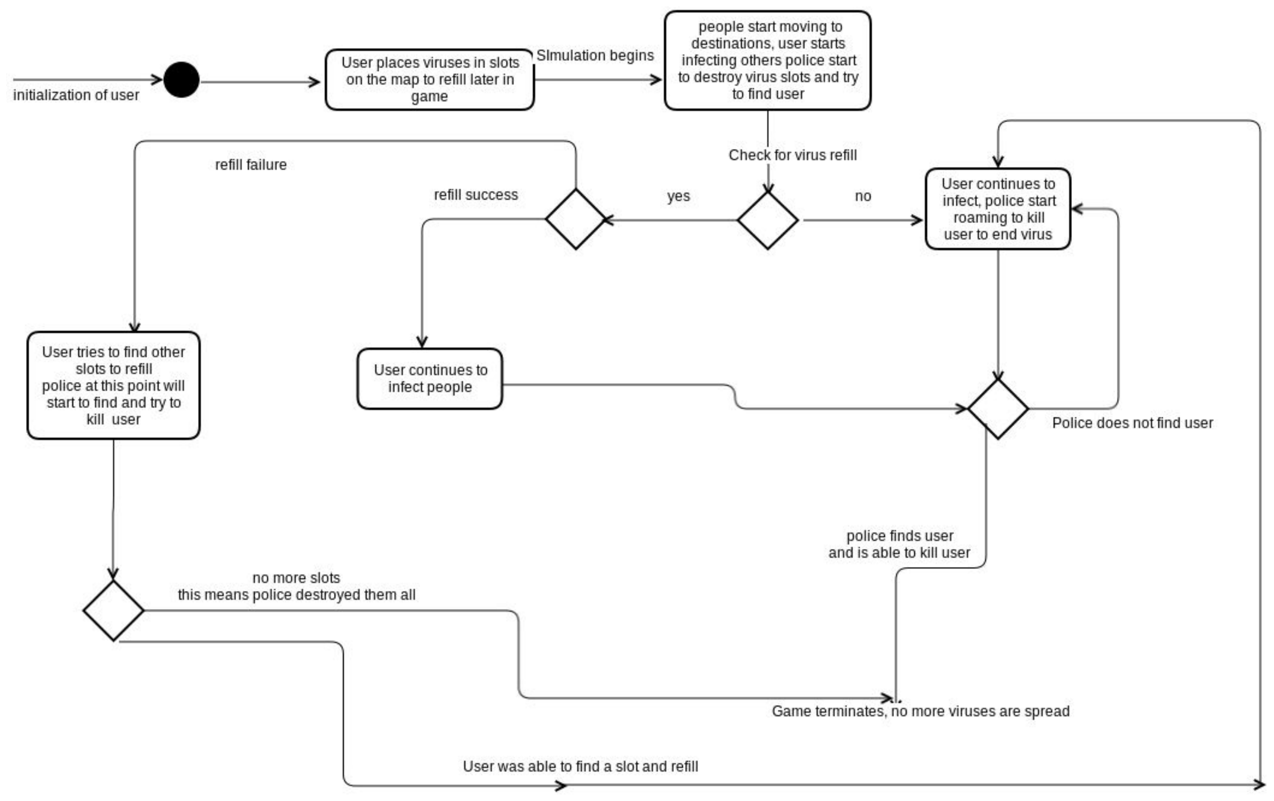
2. Overall Description

2.1 Product Perspective

There have been several games made in which the goal of the game is to spread a virus to as many people as possible. These games have mainly focused on the statistical side of population areas and how that affects virus spreading. This will be a much more direct game; in the initial phase, we will initialize the virus in a certain location, and make more locations more prone to infection. In the second phase however, we will have the player actually walking around trying to physically spread the virus to as many people as possible. So this will involve strategic, critical thinking, along with a fun, top down controller type of stealth concept.

2.2 Product Functions

The main goal of the Virus Simulation project is to simulate the spread of a virus through a city as well as to include elements of a game so that this project is fun and educational. The following shows how the functionalities of the product will come together in the following activity diagram:


initialization of user
User places viruses in slots on the map to refill later in game
SImulation begins
people start moving to destinations, user starts infecting others police start to destroy virus slots and try to find user
Check for virus refill
yes
no
refill failure
refill success
User continues to infect, police start roaming to kill user to end virus
User tries to find other slots to refill police at this point will start to find and try to kill user
User continues to infect people
Police does not find user
police finds user and is able to kill user
no more slots this means police destroyed them all
Game terminates, no more viruses are spread
User was able to find a slot and refill

2.3 Classes

Modularization

M1: World Building

- This will involve building the city in which the virus will be spread. It will also involve setting up the 2D overview map of the city in which the player initializes the virus by placing virus modules in various locations around the city. This phase will involve the Unity Editor and possibly some external tools.

M2: Player Controller

- controls the player movement in the 2D city, along with various actions the player can take.

M3: AI Controller

- This will control the various AI components, mainly people, moving about the city

M4: Virus Controller/Game Controller

- This will be utilized to spread the virus; it will take input in the form of actions by the player in which it determines if the virus is spread to something, and it will keep track of/ have data-bases of all the AI and keep track of how far the virus is spread

M5: Audio

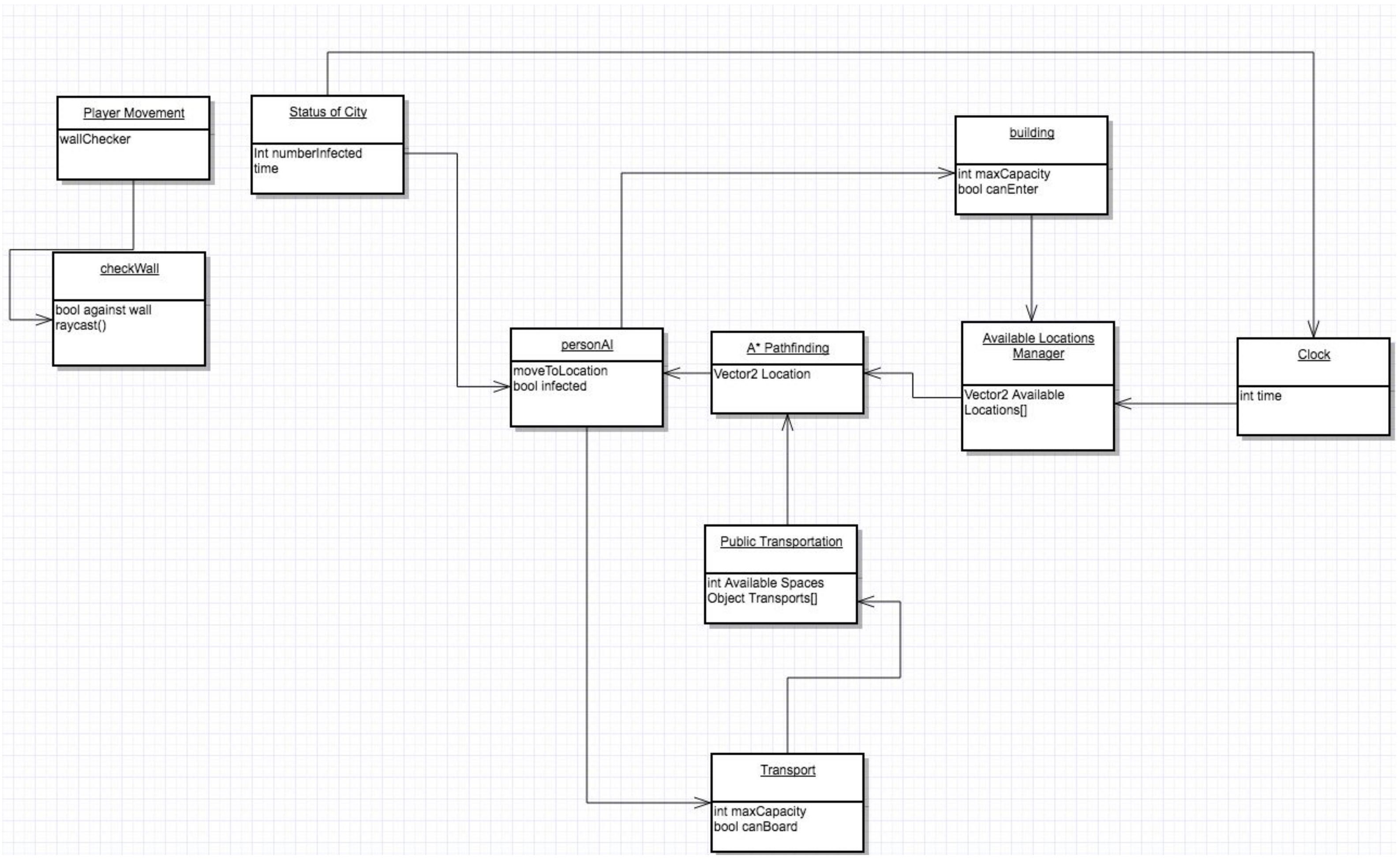
- Sound effects, music control on the various components in the game

M6: Visual Design

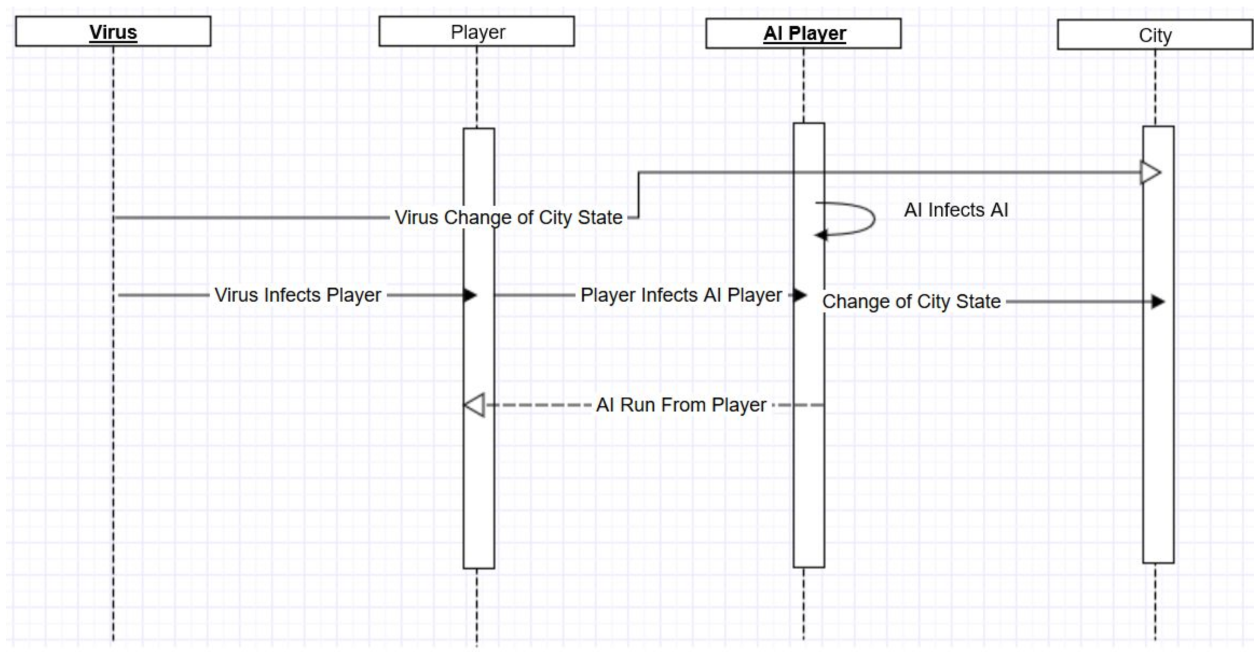
- Visually designing for the sprites, whether these are assets or designed in an external program.

M7: Menu Design

- Main Menu (separate scene)
- Mid-Game Menu
- Possibly an end game menu.



These are the classes in the program and how they relate with each other.



Sequence diagram. This shows how the player/user will start the simulation and then the simulation can play out. However the player/user can also step in and can affect the simulation causing a different outcome.

2.4 Design and Implementation Constraints

For this program we have six weeks to create a fully functioning program, so time is a major constraint. On top of this, our focus in our implementation goals will be to create working and efficient mechanics, with visual aesthetics as a secondary objective. And as this is mainly a software class, no one is explicitly expected to have artistic ability. As such, visuals will be another constraint, although they will be part of our focus towards the latter end of stage 6 implementation. Finally, as we are using a pre-built game engine framework tool, there will theoretically be constraints on tools abilities and graphical power, although this should not even come close to being a problem in a 2D arcade-like simulator. Hence why this is only a theoretical constraint.
Side note on source control constraints using world building tools - git and other source control must be integrated with a tool called YAML in order to merge changed in the World editor. Code can be merged using traditional git methods. So as of now, changes in editor are being stashed then merged using YAML-git smart merging, while base C# source code is being merged using traditional merge and branching control in git.

2.5 User Documentation

Documentation will be done with doxygen. Additionally, every C# program will be extensively documented, with every function documented and every public and private variable extentialized. There will also be an extensive document detailing in-editor states and objects.

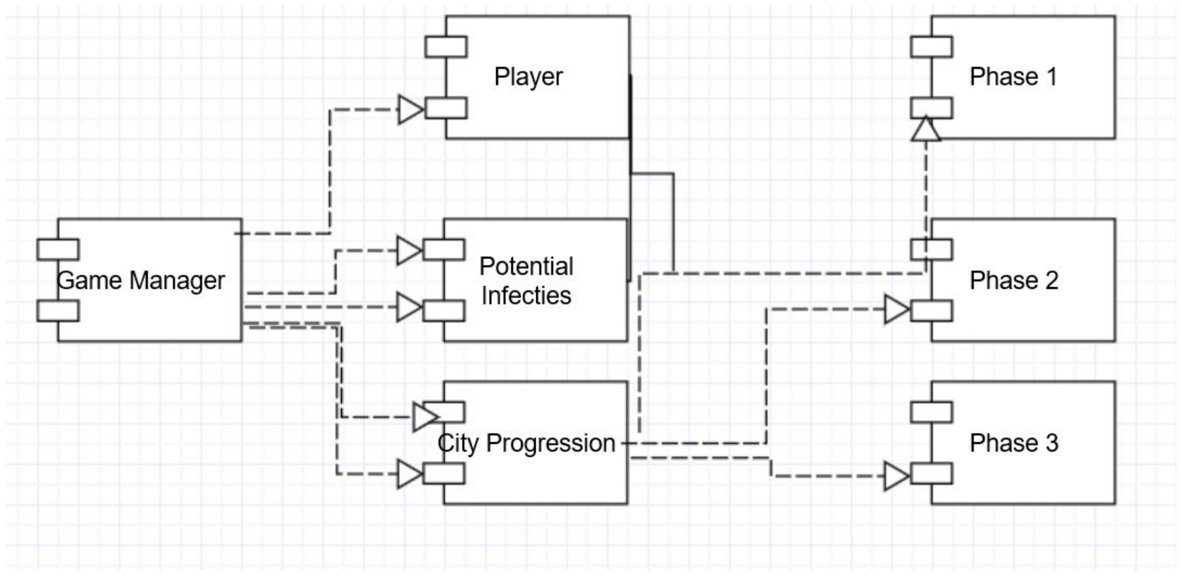
2.6 Assumptions and Dependencies

Implementation

- Unity Development Kit
  C#/JavaScript Scripting will be used for game logic and mechanics programming
  Modeling and visualization/editing software
    - Flash
    - Illustrator
    - Photoshop
    - (3D) Maya
    - (3D) Blender

  Audio Editting Software
    - Audacity
    - FL Studio

  Online Art Asset/Audio Asset resources
  Github - Source control

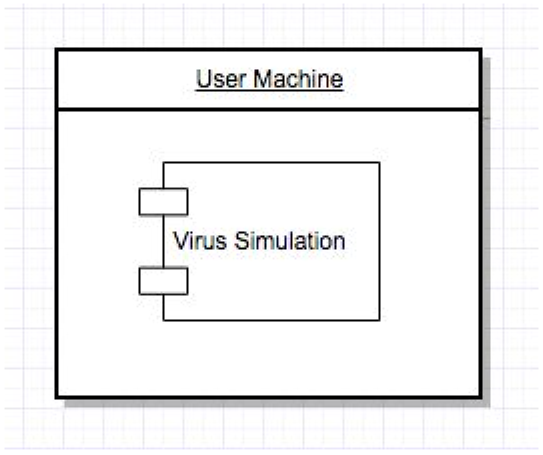


This is a component diagram for our program. The only thing it depends on is the unity engine; source code and external applications are all compiled in-engine.

3. External Interface Requirements

3.1 User Interfaces

The user will use a keyboard and mouse to interact with our program that displays on a monitor.



This is a deployment diagram for our program. Because it doesn't require any web interaction it is fairly simple. In terms of deployment UIs, it will have a simple main menu to control flow into the game, along with a visual UI in-game displaying stats.

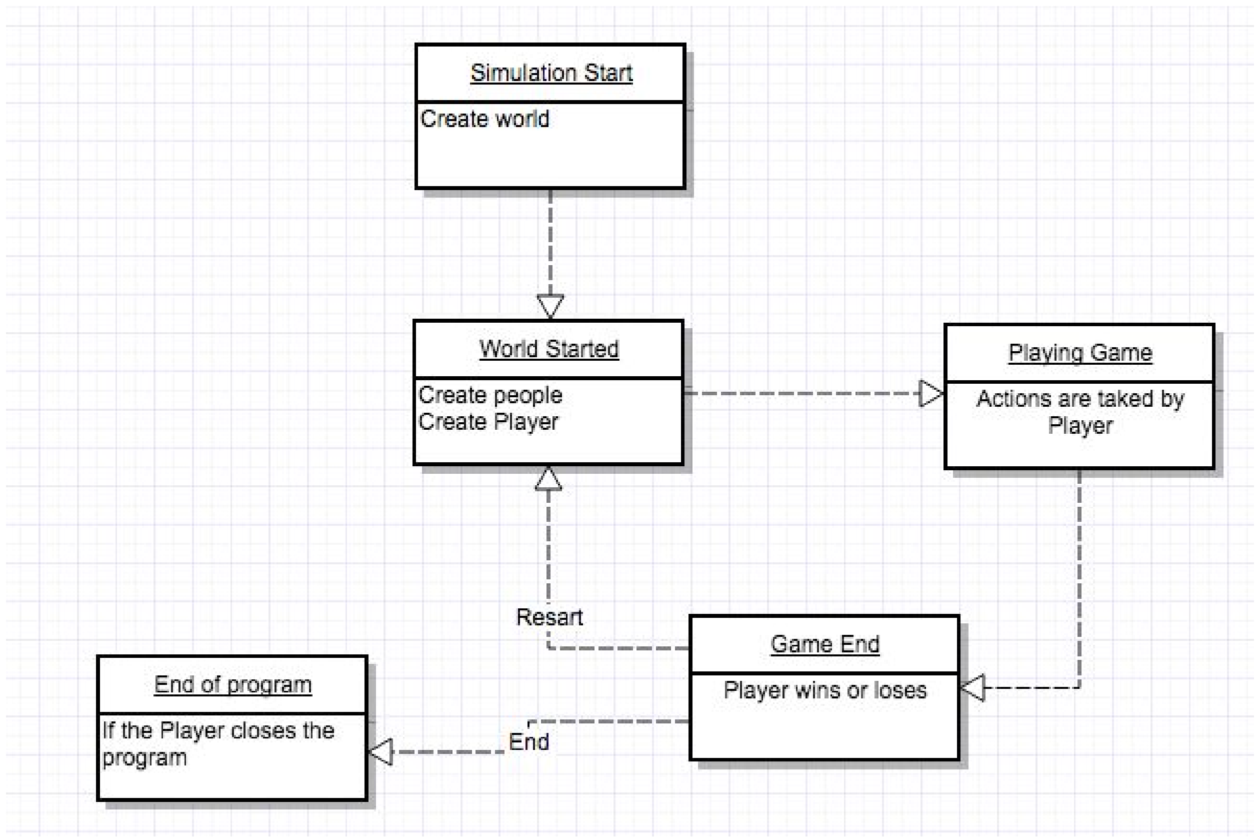
4. System Features

Mechanics

Phase 1: Top-down view: Placing several viruses in slots on the map where you will initialize the virus and then be able to refill at when the game starts. These refill stations can then be cleaned up and destroyed by enemies (police, etc.).

Phase 2: Walk around, try to sneak up on people and infect them. When you run out of vials to infect people with, refill at your stations that you stashed in phase 1.

Phase 3: The good guys come to clean up the virus and kill you! Stealth is of the utmost importance here.



This is a statechart of our program. It outlines the general progression of starting our program.